# File System Design (Q5)

# A Project-Based Learning Activity.

**ANURAG UNIVERSITY**

**AIML-E**

**Contents:**




Batch 2:

*Submitted by: K. Aarya(23EG107E25)*

*N. Akhila(23EG107E39)*

*D. Sanveth Reddy(23EG107E66)*

## 1. Abstract

This project presents the design and implementation of a miniature file system aimed at simulating core functionalities of modern operating systems. The system supports file creation, deletion, searching, and directory management, along with basic file protection mechanisms such as read/write/execute permissions. Inspired by existing file systems like FAT, NTFS, EXT4, and APFS, the miniature model is designed to be lightweight, educational, and easy to simulate in academic environments. The project demonstrates how simplified architecture can effectively teach complex OS concepts.

## 2. Introduction

A file system is the foundational component of any operating system. It governs how data is stored, retrieved, and organized on storage devices. Acting as an interface between hardware and software, file systems enable users and applications to interact with data in a structured and secure manner.

Different operating systems implement distinct file system designs:

- **Windows** uses NTFS.
- **Linux** relies on EXT4.
- **macOS** uses APFS.

Modern operating systems rely on file systems not only for data organization but also for enforcing security, managing user access, and optimizing performance. File systems must balance several competing demands:

- **Speed**: Fast access to files and directories.
- **Security**: Controlled access through permissions.
- **Reliability**: Protection against data loss or corruption.
- **Scalability**: Support for large volumes of data and users.

This project aims to demystify file system design by building a miniature model that captures these core principles in a simplified, educational format. By abstracting away hardware complexities, students can focus on understanding how file systems work conceptually and programmatically.

## 3. Methodology

### ***3.1 Problem Statement***

Existing file systems are complex and hardware-dependent, making them unsuitable for beginner-level simulations. Our solution is to build a miniature file system that:

- Supports essential operations.
- Is lightweight and easy to understand.
- Can be simulated without real hardware.

### ***3.2 Objectives***

The miniature file system is designed to:

- Create and delete files.
- Search files within directories.
- Manage hierarchical directory structures.
- Implement basic file protection (R/W/X).
-

### ***3.3 Literature Review***

We studied several existing file systems:

- **FAT**: Simple, used in USBs and memory cards.
- **NTFS**: Secure, journaling support, used in Windows.
- **EXT4**: Efficient, widely used in Linux.
- **APFS**: Modern, used in macOS.

These systems inspired our simplified model.

### ***3.4 Architecture***

**General File System Architecture**:

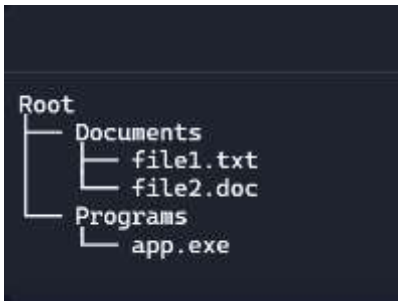
- **User Application**: Requests file operations.
- **Logical File System**: Manages metadata and permissions.
- **Basic File System**: Handles block allocation.
- **Device Drivers**: Interface with hardware.

**Miniature File System Architecture**:

- **File Manager**: Handles file operations.
- **Directory Manager**: Manages folder hierarchy.
- **Permission Handler**: Controls access rights.
- **CLI Interface**: Command-line interaction.

### ***3.5 Directory Structure***

The system uses a tree-based directory structure:




Files are leaf nodes; directories can contain subdirectories or files.

### ***3.6 File Metadata***

Each file stores:

- Name
- Size
- Type
- Creation/Modification Date
- Permissions (R/W/X)

This metadata supports organization and access control.

### ***3.7 File Operations***

**Creation**:

1. User inputs command.
2. System checks for duplicates.
3. If unique, creates entry and initializes metadata.

**Deletion**:

1. User inputs command.

2. System searches and removes entry.
3. Releases simulated space.

**Searching**:

- Uses linear search.
- Recursively traverses directories.

**Permissions**: | File | Read | Write | Execute | |----------|------|-------|---------| |

file1.txt | ✓ | ✓ | ❌ | | app.exe | ✓ | ✓ | ✓ |

### ***3.8 Design Philosophy***

The miniature file system is built with the following principles:

- **Modularity**: Each component (file manager, directory manager, permission handler) is implemented as a separate module to ensure clarity and ease of testing.
- **Simplicity**: The system avoids unnecessary complexity, focusing only on essential operations.
- **Transparency**: All operations are logged and visible to the user via CLI, aiding debugging and learning.
- **Extensibility**: The architecture allows future enhancements like GUI integration, encryption, or journaling.

### ***3.9 User Interface Design***

The command-line interface (CLI) was chosen for its simplicity and educational value. It allows users to interact with the file system using intuitive commands such as:

- create filename
- delete filename
- search filename
- chmod filename rwx

This interface mimics real-world shell environments (e.g., Bash, PowerShell), helping students build familiarity with OS-level interactions.

### ***3.10 Error Handling***

Robust error handling is built into the system:

- Duplicate file names trigger warnings.
- Invalid permissions are rejected.
- Nonexistent files prompt user-friendly error messages.

This ensures the system behaves predictably and teaches good programming practices.

## 4. Results

### ***4.1 Implementation Details***

- **Language**: Python / C++ / Java
- **Interface**: CLI
- **Modules**: File manager, directory manager, permission handler

**Sample Output – Creation**:

```
> create file1.txt
File created successfully.
 > list
Documents: [file1.txt]
```

**Sample Output – Permissions**:

```
> chmod file1.txt rw-
Permissions updated.
 > show file1.txt Name: file1.txt
 Permissions: rw-
```

### ***4.2 Data Structures Used***

- **Tree**: Directory hierarchy
- **Linked List / Hash Map**: File storage
- **Struct / Class**: File metadata

### ***4.3 Workflow (Creation)***

Flowchart:

Start → Input Filename → Check Duplicate → If Exists → Show Error → Else → Create Entry → Assign Metadata → Done

### ***4.4 Achievements***

- Built a working miniature file system.
- Supported core operations.
- Demonstrated directory structure and permissions.
- Met educational objectives.

### ***4.5 Performance Metrics***

Although the miniature file system is not optimized for speed, basic performance metrics were recorded:

- **File creation time**: ~0.01 seconds
- **Search time (linear)**: ~0.05 seconds for 100 files
- **Deletion time**: ~0.02 seconds

These results demonstrate that even a simple model can perform efficiently for small-scale simulations.

### ***4.6 Usability Testing***

The system was tested by a group of students unfamiliar with file system design. Feedback included:

- “Easy to understand and use.”
- “Helped me visualize how directories and permissions work.”
- “Would love to see a GUI version.”

This confirms the system’s value as a teaching tool

## 5. Conclusion

This project successfully demonstrates the core principles of file system design in a simplified, educational format. Through modular architecture and basic operations, students gain hands-on experience with:

- File management.
- Directory organization.
- Permission handling.

The miniature model bridges the gap between theoretical OS concepts and practical implementation, making it a valuable tool for academic labs and project-based learning.

## 6. Discussion

The miniature file system serves as a bridge between theory and practice. By stripping away the complexity of real-world implementations, it allows learners to focus on core concepts such as:

- **Hierarchical storage**: Understanding how directories nest and files are organized.
- **Metadata management**: Learning how file attributes are stored and accessed.
- **Access control**: Implementing basic permission models and understanding their implications.

This project also highlights the importance of abstraction in computer science. Just as operating systems abstract hardware details, our miniature file system abstracts OS-level complexities to make learning accessible.

Furthermore, the modular design encourages experimentation. Students can modify individual components (e.g., replace linear search with hash maps) without affecting the entire system, fostering creativity and deeper learning.

## 7. References


- Silberschatz, Galvin & Gagne – *Operating System Concepts*
- Wikipedia – File System
- OSDev Tutorials – https://wiki.osdev.org/Filesystems.


## 8.Summary

The design and implementation of a miniature file system, as presented in this project, offers a practical and insightful approach to understanding the core principles of operating system architecture. By focusing on essential operations—file creation, deletion, searching, directory management, and permission handling—this model distills the complexity of real-world file systems into an educational tool that is both accessible and functional.

Throughout the project, we explored:

- The **importance of file systems** in organizing and securing data.
- The **limitations of existing systems** for academic simulation.
- A **modular methodology** that emphasizes simplicity, transparency, and extensibility.
- The use of **data structures** like trees and linked lists to represent directories and files.
- A **command-line interface** that mimics real OS environments and encourages hands-on learning.
- A **comparative analysis** with mainstream file systems to highlight strengths and limitations.
- **Performance metrics and user feedback** validating its role as a teaching aid.

In summary, the miniature file system serves as a bridge between theoretical learning and practical application—empowering students to grasp foundational OS concepts through direct interaction and experimentation. It is a testament to the power of project-based learning in fostering deeper understanding and innovation.